IN THE SPECIFICATION

Please amend paragraph [0001] as follows:

[0001] This application claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/464,283, filed December 15, 1999, by Laurence B. Boucher et al., now U.S. Patent No. 6,427,173, which in turn claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/439,603, filed November 12, 1999, by Laurence B. Boucher et al., now U.S. Patent No. 6,247,060, which in turn claims the benefit under 35 U.S.C. § 119(e)(1) of the Provisional Application Serial No. 60/061,809, filed on October 14, 1997. This application also claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/384,792, filed August 27, 1999, now U.S. Patent No. 6,434,620, which in turn claims the benefit under 35 U.S.C. § 119(e)(1) of the Provisional Application Serial No. 60/098,296, filed August 27, 1998.

Please amend paragraph [0002] as follows:

[0002] This application also claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/067,544, filed April 27, 1998, now U.S. Patent No. 6,226,680. This application also claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/416,925, filed October 13, 1999, now U.S. Patent No. 6,470,415. The subject matter of all the above-identified patent applications, and of the two above-identified provisional applications, is incorporated by reference herein.

Please amend paragraph [0025]as follows:

[0025] Figure 3 is a flow chart that illustrates a method in accordance with an embodiment of the invention. Processor 210, as it proceeds down the thread, places a set of multiple (in this case two) DMA commands into SRAM 213. The first DMA command is to move first portion of data 219 from address X1 in host storage 204 to address Z1 in local memory 207. After placing the first DMA command into SRAM 213 via lines 222 and SRAM controller 212, processor 210 causes a value that points to the first DMA command in SRAM 213 to be pushed (step 300) onto DMA command queue 217. Processor 210 does this by first writing a queue identifier into a queue control register (not shown) in queue manager 209. This queue identifier indicates the queue onto which a subsequently written 32-bit queue data value is to be pushed. Processor **210** then writes a 32-bit queue data value onto a queue data register (not shown) in queue manager 209. The 32-bit queue data value contains: a 16-bit address where the DMA command is located in SRAM 213, a 5-bit termination queue identifier, an 8-bit termination value to be pushed onto the queue identified by the termination queue identifier upon completion of the DMA command, a one-bit DMA chain bit value, a one-bit dummy DMA value, and one unused bit. If the DMA chain bit is set, this indicates ~~the~~ **that** when the DMA controller has completed the DMA command, that the DMA controller is not to push a termination value onto any queue. If the dummy DMA bit is set, this indicates that the DMA controller is not to perform a DMA command per se but is nevertheless to push the 8-bit termination value onto the queue identified by the 5-bit termination queue identifier.

Please amend paragraph [0030] as follows:

[0030] Only after both portions of data 219 and 220 are present on the network interface device 200, does processor 210 branch to the instructions that cause the data of portions 219 and 220 to be output from the network interface device

in the form of a data ~~paylaod~~ **payload** of a network communcation. To ensure that all of the necessary data portions have been moved to the network interface device 200, processor 210 need not monitor the complete status of all the DMA commands involved as in the prior art example of Figure 1. Rather, processor 210 polls a 32-bit queue-out-ready register (not shown) in queue manager 209. Each bit in the queue-out-ready register indicates whether there is a value present in a corresponding queue. Processor 210 therefore polls the queue-out-ready register, determines that the bit for the DMA command complete queue is set, and then pops the DMA command complete queue 218. Processor 210 repeats this process until the value popped off the DMA command complete queue 218 is the value indicative of the last DMA command in the set of DMA commands. When the value popped off the DMA command complete queue is the value indicative of the last DMA command in the set of DMA commands, then it is known that all the proceeding DMA commands in the set have been executed.